Gesture Based UI Development

John Shields - G00348436

GitHub Repository URL: **https://github.com/johnshields/Eithne-Voice-Assistant**

Introduction

Originally for this project, a Virtual Reality detective game was the first choice. This game would have been developed with Unity and the Oculus Quest. Unfortunately, due to computer hardware limitations, it was not feasible to develop a sufficient game. Many attempts were made to get the game set up, but even basic setups took hours. Being a hectic time during the college year, this could not carry on. Time is precious; therefore, improvisation had to be made. The project's goal was then altered to be a Voice Assistant in Python with skills enhanced by AI technologies.

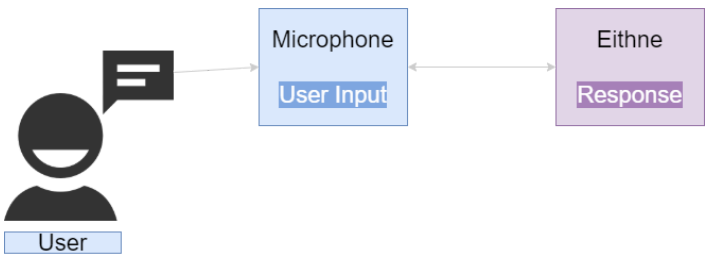
Voice Assistants

Voice assistants are software programs that have been designed to assist users with basic tasks using an inbuilt natural language user interface. The most common VAs embedded in smartphones or dedicated home speakers are Apple's Siri, Amazon's Alexa, Microsoft's Cortana, and Google's Assistant. Users can use voice commands to ask their assistants questions, monitor home automation systems and media playback, and handle other essential tasks, including skills such as application control and information gathering. [1] & [2]

Purpose of the Application

The application designed is a Voice Assistant by the name of Eithne. Eithne takes in a voice command from a user and responds to the User depending on the commands (Figure Below). The User's voice input is done by a Speech Recognition API, and Eithne's voice comes from the Google Text-To-Speech API. The skills/features Eithne performs are shown with figures in the ***Features*** subsection.

User Input and Response System



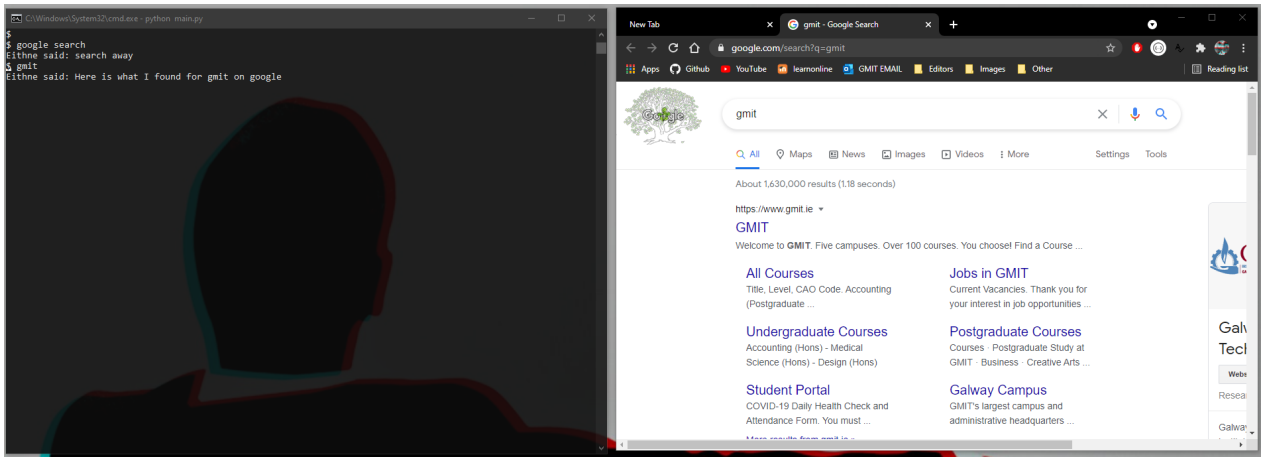
Features

Eithne is programmed to do the following features:

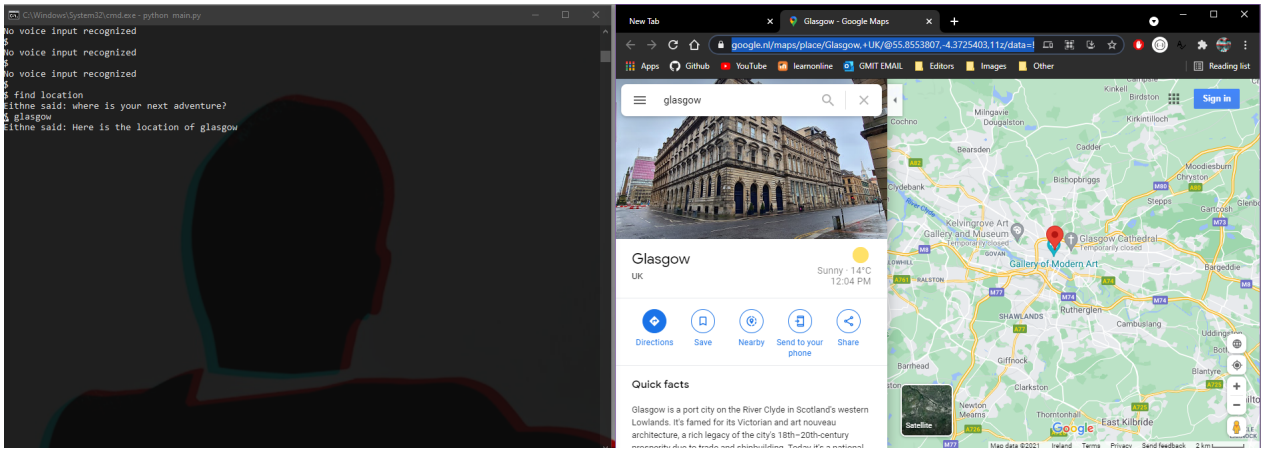
- Google Search
- Google Maps
- Wikipedia Summaries
- YouTube Queries
- Open any Website with a `dot com`
- Historical Events that happened Today from numbersapi.com/day/month/date

See the figures below for demonstrations of how the User interacts with Eithne's integrated features.

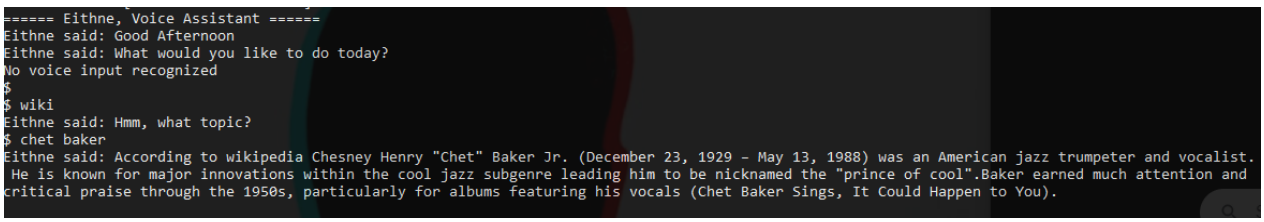
Google Search Feature



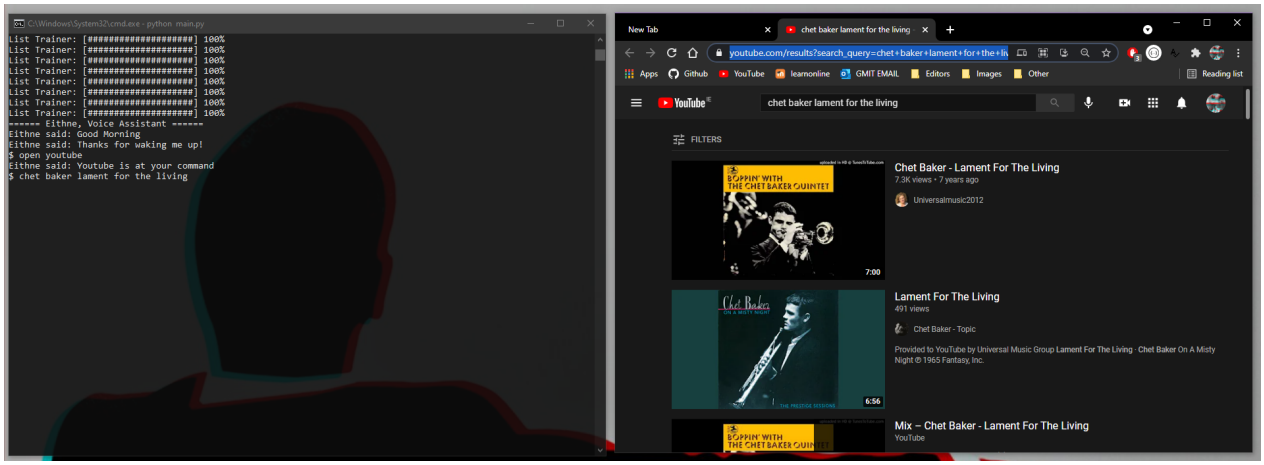
Google Maps Feature



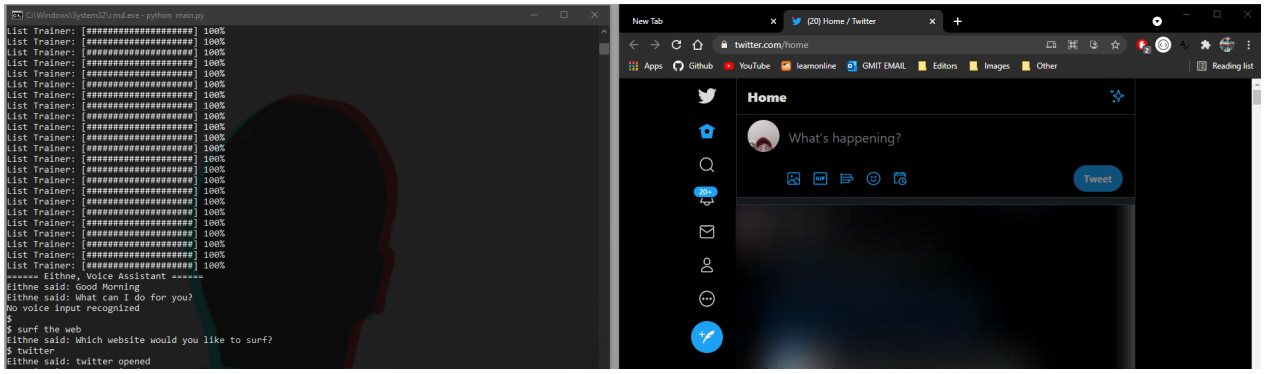
Wikipedia Summaries Feature



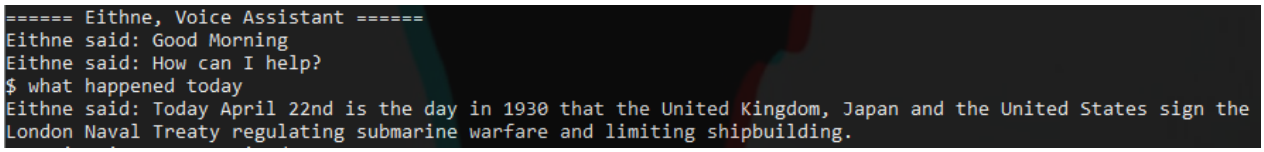
YouTube Queries Feature



Website Feature - Twitter



Historical Events Feature



Gestures of the Application

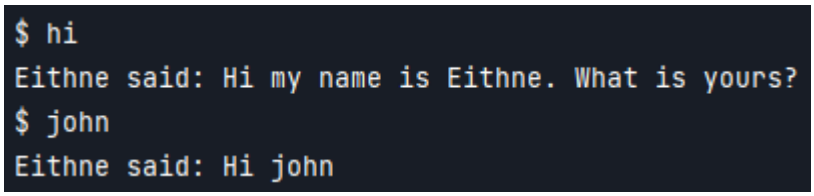
The application is entirely controlled by voice. Meaning the gestures implemented are user commands and also what Eithne says back to the User. The User controls Eithne by having almost a conversation for each command as they always get a response no matter what the User says. Being a voice assistant, the commands had to be pretty open so that they would come naturally to the User. In order to achieve this in `user_phrases.py` a function was created to return many possible commands a user could say for the features above. Also, there are simple commands, for example, `Hi`, `Hello`, `Hey` all return with a response by Eithne asking for the User's name.
For Eithne's responses, a machine learning (ML) mechanism was used to get different responses for each command. The following are the programmed commands, but the User is not limited to just these as the ML mechanism can understand similar commands to the ones said and still give back a response. The ML mechanism will be discussed in far more detail in ***Architecture of the Application***.

For the features, Google Search, Google Maps, Wikipedia, YouTube Queries, and Website, the User is required to say two commands. One for activating the feature and one for controlling what they want the feature to do. There are figures under the commands for these features for further explanation.

Gesture Commands

User wants to know the VA's name or wants to say hello.	
User Input	**Response**
What is your name?	My name is Eithne. What is yours?
Who are you?	I'm Eithne. Who are you?
Hi	Hi my name is Eithne. What is yours?
Hello	Hello, I'm Eithne. You?
Hey	Hey yourself! Eithne here. What's your name?

After the User says, their name Eithne will say Hi with the User's name (Figure below).



User wants to know what Eithne can do.	
User Input	**Response**
Tell me about yourself	I am a voice assistant named Eithne…
Talk about yourself	^^
What can you do?	^^
About	^^

User wants to thank Eithne.	
User Input	**Response**
Thank you	You're welcome!
Thanks	Don't mention it!
Sound	No bother!
Cheers	No problem!

User wants to do a Google Search.	
User Input	**Response**
Search	What would you like to search for?
Do a Search	Google is loaded for searching!
Google search	Search away!

User wants to do a Google Search.	
Open google	Google is waiting your command!

```
$ google search
Eithne said: Search away!
$ dogs
Eithne said: Here is what I found for dogs on Google
```

User wants to find a location.	
User Input	**Response**
Location	What is the location?
Maps	What place?
Find location	Where is your next adventure?
Where can I find	Find what?
Open Maps	Maps is at your command!

```
$ location
Eithne said: What is the location?
$ galway
Eithne said: Here is the location of galway
```

User wants to use Wikipedia.	
User Input	**Response**
Wikipedia	What would you like to know more about?
Wiki	Hmm, what topic?

```
$ wikipedia
Eithne said: What would you like to know more about?
$ the grand budapest hotel
Eithne said: According to wikipedia The Grand Budapest Hotel is a 2014 comedy-drama film
```

User wants to know what happened today in history.	
User Input	**Response**
History	Historical event that happened today…
What happened today?	^^
History of today	^^

User wants to watch a video on YouTube.	
User Input	**Response**
YouTube	YouTube is waiting for a query!
Video	What video?
I would like to watch a Video	Which one?
Open YouTube	YouTube is at your command!

```
$ i would like to watch a video
Eithne said: Which one?
$ arrival opening scene
Eithne said: Here are videos for arrival opening scene on YouTube
```

User wants to surf a website.	
User Input	**Response**
Website	Which website?
Surf the web	Which website would you like to surf?
Internet	What would you like to see?
Online	Online. Waiting on your command!

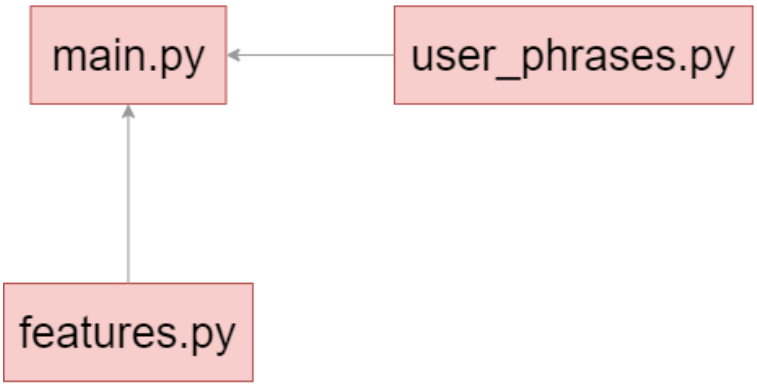
```
$ online
Eithne said: Online. Waiting on your command!
$ github
Eithne said: github opened
```

User wants Eithne to stop listening/turn off.	
User Input	**Response**
Turn off	Farewell
Stop listening	Goodbye
Exit	Good Luck
Quit	So long

Architecture of Application

The application's design consists of three controllers. `main.py` `user_phrases.py` and `features.py` (Figure Below). Main handles the integrations of user voice input, Eithne's responses and also trains a chatbot for Eithne's responses. User Phrases sets up initial commands for a user to say. Feature handles opening the browser for Websites, Google Search, Maps and YouTube, Wikipedia, and Historical events requests.

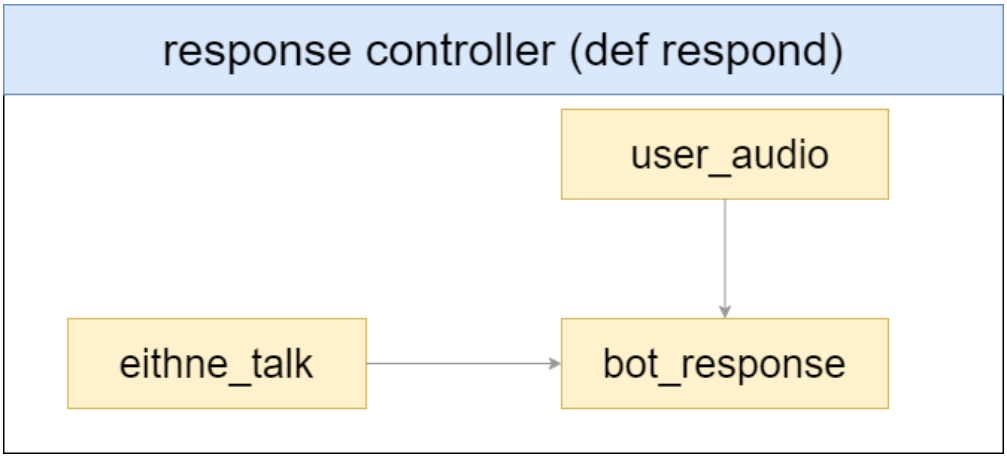
Controllers Diagram



User Interaction

The User's main interaction with Eithne is controlled by the functions `user_said` from `user_phrases.py` , `features.py` , `eithne_talk` , `bot_response` and `user_audio` from `main.py` .

Response Controller



The User's input is taken in by the function `user_audio` (Code below). This creates a microphone instance for the User. This is then set up to listen and recognizes speech. It tries to listen for user input if there was none or it was unintelligible. The User gets a response from a print statement and an else condition from the function `respond` that makes Eithne alert the User for invalid inputs.

Function for recognizing User Input

```
def user_audio(ask=''):
    # Create a microphone instance.
    with sr.Microphone() as source:
        # If the command requires 2 inputs from the user.
        if ask:
            eithne_talk(ask)

        # Listen for user input.
        audio = r.listen(source)
        user_input = ''

        try:
            # Try to listen for user input...
            user_input = r.recognize_google(audio, language='en')
        except sr.UnknownValueError:
            # If nothing was said or speech was 'unintelligible'.
            print('No voice input recognized')
        except sr.RequestError:
            # Speech Recognition is unreachable or unresponsive.
            eithne_talk('Sorry, it appears that the speech recognition service is down.')
            exit()
        print(f'$ {user_input.lower()}')
        return user_input.lower()  # Return the user's input.
```

Code of user commands for greeting Eithne

```
def user_said(phrase):
    p = phrase
    # User wants to know the VA's name.
    if 'what is your name' in p or 'who are you' in p or 'hi' in p or 'hello' in p or 'hey' in p:
        p = 'name'
```

Function for interactions between Eithne and a user

```
def respond(user_input):
    # Allow user to ask for VA's name then say 'Hi' with the user's said name.
    if 'name' in user_said(user_input):
        name = user_audio(bot_response(user_input))  # Get response from trained bot.
        eithne_talk(f'Hi {name}')
```

Code for invalid responses

```
else:
    n = ['Sorry I cannot process that command', 'Did you say something?',
         'Are you alright?', 'Unreachable command', "Sorry, what was that?", "Does not compute"]
    message = n[random.randint(0, len(n) - 1)]
    eithne_talk(message)
```

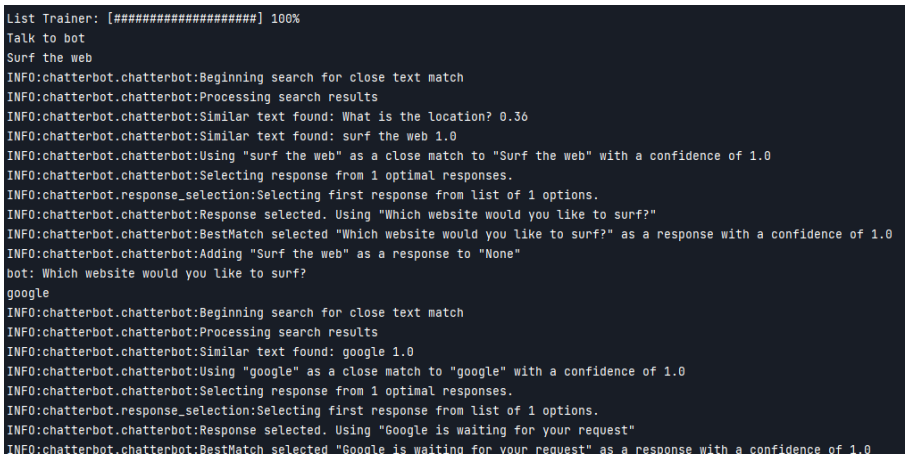
Responses

The majority of Eithne's responses are decided by a machine learning engine called ChatterBot. ChatterBot is used to train Eithne to respond to user commands for features in the application. The bot is designed to respond to multiple commands for each feature. The Logic Adapter used for this functionality is `BestMatch`. The code below shows how the bot is trained with possible user commands and set responses.

```
# Train the bot.
trainer = ListTrainer(eithne_bot)
# Response to a google search cmd.
trainer.train(["search", "What would you like to search for?"])
trainer.train(["do a search", "Google is loaded for searching!"])
trainer.train(["google", "Google is waiting for your request!"])
trainer.train(["google search", "Search away!"])
trainer.train(["open google", "Google is waiting your command!"])
```

ChatterBot was initially tested with these commands and responded through a command-line interaction. This testing was mainly to test how the bot learns with each command passed in. The figure below is a command-line interaction with the bot that shows a log output of how it learns as commands are given.

Testing Bot's responses



The learned responses are stored in a SQLite database to learn from the User and improve the application continuously.

The voiced responses are handled by the function `bot_response`. This function is used to return a response for Eithne to say depending on the command the User said.

Function to take in a user command and return a response from the bot

```
def bot_response(cmd):
    response = eithne_bot.get_response(cmd)
    return str(response)
```

Eithne's vocals are handled by Google Text-To-Speech. The standard voice suited the application well and it can say sentences in a very understandable matter. The code below shows how Eithne can talk with the Text-To-Speech software.

Function to use Google Text-To-Speech for Eithne's voice

```
def eithne_talk(audio_string):
    # Set up Eithne's vocals.
    talk = gTTS(text=audio_string, lang='en')
    ran_num = random.randint(1, 100000)
    # Set to random mp3 file and save.
    audio_file = 'audio-' + str(ran_num) + '.mp3'
    talk.save(audio_file)
    # Play what Eithne said then remove the mp3.
    playsound.playsound(audio_file)
    print(f'Eithne said: {audio_string}')
    os.remove(audio_file)
```

Features of the Application

As said above, the Google Search, Google Maps, Wikipedia, YouTube Queries, and Website features have simple functionality to perform for the users. The response time for these functions is generally relatively quick. The only one with a slight delay is the Wikipedia features as Google Text-To-Speech has to record the Wikipedia summary and then play the recording to the user. Below are two of the functions of said features.

Function to take in user input and do a google search.

```
def google(search):
    # Set up URL.
    url = f'https://google.com/search?q={search}'
    wb.get().open(url)  # Open browser with URL
```

Function to get Historical Events

```
def on_this_day():
    d = datetime.today().strftime('%d')
    m = datetime.today().strftime('%m')
    # Set up URL.
    url = f'http://numbersapi.com/{m}/{d}/date'
    # Do GET request.
    resp = req.get(url)
    return resp.text  # return the response body's text
```

Issues encountered

Apart from the issues with the original Virtual Reality game idea, Eithne was a much more achievable goal. It was a perfect idea for improvisation as it was tough to know if the VR game would have even had its basic functionality before the project's deadline.

Python's Speech Recognition API requires a strong internet connection, meaning at times, it can take a while to recognize user input if there is a poor connection. This issue did cause some setbacks but not enough to disturb the development of the application. ChatterBot did take some time to train and test for accurate responses for user input. However, in the end, ChatterBot proved to be an excellent tool to have in the application.

The application has been thoroughly tested to make sure every possible issue a user could experience is very limited or none at all. The Test Sheet is located here.

Conclusions & Recommendations

Being a voice-focused application, the hardware scope used in development was small, consisting of only a Microphone. A Raspberry Pi would have been a great addition but unfortunately, due to the Voice Assistant idea being an improvisation. There was not enough time to acquire one before the project's deadline. All in all, I believe the application was a success, and I am pleased with the final product. Mixing gestures with software and hardware is interesting to develop and convenient for the User. I am delighted I chose Python for the application as, before this, I have never used it immensely. I can now add Python to my skills and use it much more in the future as I am now a lot more comfortable with it. Having the issue with the VR game made me realize I have to do far more research before thinking of an idea for a project with an open brief. Many hours were wasted trying to get the game set up and have it achieve something. All this time wasted could have been avoided if I researched the computer hardware expectations wholly to develop a VR game instead of jumping headfirst into development and assuming everything would fall into place. Thus a very important lesson was learnt with this project.

References

- Intelligent Personal Assistants Automated Personal Assistants
- Alexa, Siri, Cortana, and More: An Introduction to Voice Assistants
- Build A Python Speech Assistant App
- How to build your own AI personal assistant using Python
- The Ultimate Guide To Speech Recognition With Python
- ChatterBot Documentation

Relevant Libraries used:

- Speech Recognition ~= 3.8.1
- Google Text-To-Speech ~= 2.2.2
- Play Sound ~= 1.2.2
- ChatterBot ~= 1.0.4
- PyAudio ~= 0.2.11